Algoritmos y Estructuras de Datos II

Parcial 25-04: TEMA “A”

Ejercicio 1: Listado de personajes

Se pretende diseñar una pequeña herramienta que permita cargar y analizar los datos de una partida de juegos de rol con 10 personajes posibles (participan 8). Para tal fin se elaboró un archivo que contiene la información de cada personaje teniendo en cuenta:

Nombre (**`'name'`**, 5 caracteres), Vitalidad (**`‘Life’`**, `[1, 100]`), Fuerza (**`‘strength’`**, `[1, 10]`), Agilidad (**`‘agility’`**, `[1, 10]`) y alineación (**`‘alignment’`**, `'good'` o `'evil'`).

En el directorio del ejercicio se encuentran los siguientes archivos:

Archivo	***Descripción***
`main.c`	Contiene la función principal del programa
`character.h`	Declaraciones relativas a la estructura de los personajes junto con las funciones de carga y escritura de datos.
`character.c`	Implementaciones **incompletas** de las funciones
`array_helpers.h`	Declaraciones / prototipos de las funciones que manejan la tabla de personajes
`array_helpers.c`	Implementaciones **incompletas** de las funciones que manejan el arreglo

Abrir el archivo **`input/valid.in`** para ver cómo se estructuran los datos.

Cada línea de datos tiene la estructura que se describe a continuación. Inicialmente dos caracteres que indican:

- **Tipo de personaje:** Se representa con un caracter que es uno entre **`'a'`**, **`'p'`**, **`'t'`**, **`'m'`** que denotan el personaje de tipo ágil, personaje de fuerza física, personaje resistente (tanque) y personaje mágico respectivamente.
- **Alineación:** Es un carácter que representa el alineamiento del personaje, ya sea "bueno" o "malo". Debe ser la letra **`'g'`** (bueno) o **`'e'`** (malo).

estos dos primeros caracteres estarán encerrados entre corchetes, por ejemplo **`'[a g]'`**. Luego seguirá un espacio y a continuación una secuencia de 5 caracteres que denotan el:

- **Nombre del personaje:** Se representa con exactamente 5 caracteres. Por ejemplo, **`'Eddie'`**.

Luego sigue un espacio y a continuación los datos restantes son una lista con los atributos de cada personaje separados por coma (,):

- **Vida**: Es un valor numérico entero que representa la cantidad de vida del personaje. Está precedido por la palabra **`'life:'`** y seguido de una coma. El valor debe estar en el rango de 0 a 100. Por ejemplo, **`'life: 80,'`**.

- **Fuerza:** Es un valor numérico entero que representa la fuerza del personaje. Está precedido por la palabra **`'strength:'`** y seguido de una coma. El valor debe estar en el rango de 1 a 10. Por ejemplo, **`'strength: 6,'`**.
- **Agilidad:** Es un valor numérico entero que representa la agilidad del personaje. Está precedido por la palabra **`'agility:'`** y seguido de una coma. El valor debe estar en el rango de 1 a 10. Por ejemplo, **`'agility: 8,'`**.

Lo que quiere decir que cada fila tiene el siguiente formato:

```
[<a | p | t | m> <g | e>] <Name> life: <[0-100]>, strength: <[0-10]>, agility: <[0-10]>
```

Ejemplo:

```
[p e] Nadia life: 39, strength: 5, agility: 3
```

Consideraciones:

- El nombre tiene exactamente 5 caracteres.
- Todas las cantidades son enteras, positivas.
- El valor de mercado es un entero positivo.
- Se pueden repetir los nombres (esto no es parte del ejercicio)

El ejercicio consiste en completar el procedimiento de carga de datos en los archivos **`array_helpers.c`**, **`array_helpers.h`** y **`character.c`**. Recordar que el programa tiene que ser *robusto*, es decir, debe tener un comportamiento bien definido para los casos en que la entrada no tenga el formato esperado.

Tener en cuenta que el registro el tipo **`Character`** (definido en **`character.h`**), tiene dos particularidades:

1. Tiene un campo '**`bool alive`**' que debe ser '**`TRUE`**' si la cantidad de vida (**`life`**) del personaje es mayor que **`0`**.
2. '**`alignment_t alignment`**' tiene como valor uno de los valores de la enumeración **`alignment_t`** (definida también en **`character.h`**) y debe ser '**`good`**' si el valor del registro es '**`g`**' y '**`evil`**' si el valor es '**`e`**'.

Una vez completada la lectura de datos se puede verificar si la carga funciona compilando,

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -g -c array_helpers.c character.c main.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 array_helpers.o character.o main.o -o
load_characters
```

y luego ejecutar

```
$ ./load_characters input/valid.in
```

Ejercicio 2: Análisis de los datos

Completar la siguiente función definida en **`array_helpers:`**

```
float array_alive_mean_life(CharacterList array);
```

Calcula la media de vida de los personajes vivos. O más en detalle: Calcula la suma de las vidas de todos los personajes vivos, divide la suma entre el número de personajes vivos y devuelve el resultado. Si no hay personajes vivos, devuelve 0.

Ejemplo:

Si vemos el contenido de **'inputs/valid_2.in'**

```
[a g] Alice life: 83, strength: 7, agility: 5
[a e] Lenny life: 71, strength: 4, agility: 8
[p g] Mario life: 92, strength: 9, agility: 6
[p e] Nadia life: 0, strength: 5, agility: 3
[t g] Bella life: 25, strength: 2, agility: 6
[t e] Oscar life: 95, strength: 10, agility: 4
[m g] Yoshi life: 0, strength: 7, agility: 10
[m e] Frank life: 68, strength: 8, agility: 7
```

Vemos que Nadia y Yoshi **no** están vivos, por lo que el promedio de vida de los vivos se calcula como:

```
(83 + 71 + 92 + 25 +95 + 68) / 6 = 71
```

Finalmente modificar el archivo **`main.c`** para que se muestre **`array_alive_mean_life`** por consola.

Hints!

1. Debug.
2. Se provee un ejemplo para cada caso que debe fallar (inputs/invalid_*.in), y dos ejemplos de archivos validos (inputs/valid_*.in).
3. Lean todo el código provisto.
4. Usen las constantes declaradas en los .h y completen donde corresponda.
5. Enserio usen debugging (gdb).
6. Relájense es solo un examen, la vida continúa mañana.